06

Para saber más: ¿Qué hay de las clases DAO?

En algunos proyectos Java, dependiendo de la tecnología elegida, es común encontrar clases que siguen el patrón **DAO**, usado para aislar el acceso a los datos. Sin embargo, en este curso usaremos otro patrón, conocido como **Repositorio**.

Pero entonces pueden surgir algunas preguntas: ¿cuál es la diferencia entre los dos enfoques y por qué esta elección?

Patrón DAO

El patrón de diseño DAO, también conocido como **Data Access Object**, se utiliza para la persistencia de datos, donde su objetivo principal es separar las reglas de negocio de las reglas de acceso a la base de datos. En las clases que siguen este patrón, aislamos todos los códigos que se ocupan de conexiones, comandos SQL y funciones directas a la base de datos, para que dichos códigos no se esparzan a otras partes de la aplicación, algo que puede dificultar el mantenimiento del código y también el intercambio de tecnologías y del mecanismo de persistencia.

Implementación

Supongamos que tenemos una tabla de productos en nuestra base de datos. La implementación del patrón DAO sería la siguiente:

Primero, será necesario crear una clase básica de dominio `Producto` :

```
public class Producto {
    private Long id;
    private String nombre;
    private BigDecimal precio;
    private String descripcion;

    // constructores, getters y setters
}
```

COPIA EL CÓDIGO

A continuación, necesitaríamos crear la clase `ProductoDao`, que proporciona operaciones de persistencia para la clase de dominio `Producto`:

```
public class ProductoDao {

    private final EntityManager entityManager;

    public ProductoDao(EntityManager entityManager) {
        this.entityManager = entityManager;
    }

    public void create(Producto producto) {
        entityManager.persist(producto);
    }

    public Producto read(Long id) {
        return entityManager.find(Producto.class, id);
    }

    public void update(Producto producto) {
        entityManger.merge(producto);
    }
```

```
    public void remove(Producto producto) {
        entityManger.remove(producto);
    }

}
```

COPIA EL CÓDIGO

En el ejemplo anterior, se utilizó JPA como tecnología de persistencia de datos de la aplicación.

Padrón Repository

Según el famoso libro *Domain-Driven Design* de Eric Evans:

> El repositorio es un mecanismo para encapsular el almacenamiento, recuperación y comportamiento de búsqueda, que emula una colección de objetos.

En pocas palabras, un repositorio también maneja datos y oculta consultas similares a DAO. Sin embargo, se encuentra en un nivel más alto, más cerca de la lógica de negocio de una aplicación. Un repositorio está vinculado a la regla de negocio de la aplicación y está asociado con el agregado de sus objetos de negocio, devolviéndolos cuando es necesario.

Pero debemos estar atentos, porque al igual que en el patrón DAO, las reglas de negocio que están involucradas con el procesamiento de información no deben estar presentes en los repositorios. Los repositorios no deben tener la responsabilidad de tomar decisiones, aplicar algoritmos de transformación de datos o brindar servicios directamente a otras capas o módulos de la aplicación. Mapear entidades de dominio y proporcionar funcionalidades de aplicación son responsabilidades muy diferentes.

Un repositorio se encuentra entre las reglas de negocio y la capa de persistencia:

1. Proporciona una interfaz para las reglas comerciales donde se accede a los objetos como una colección;
2. Utiliza la capa de persistencia para escribir y recuperar datos necesarios para persistir y recuperar objetos de negocio.

Por lo tanto, incluso es posible utilizar uno o más DAOs en un repositorio.

¿Por qué el padrón repositorio en lugar de DAO usando Spring?

El patrón de repositorio fomenta un diseño orientado al dominio, lo que proporciona una comprensión más sencilla del dominio y la estructura de datos. Además, al usar el repositorio de Spring, no tenemos que preocuparnos por usar la API de JPA directamente, simplemente creando los métodos, que Spring crea la implementación en tiempo de ejecución, lo que hace que el código sea mucho más simple, pequeño y legible.